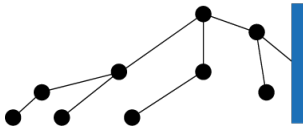
POLITECNICO
MILANO 1863



EVEREST

FLASHLIGHT Workshop @ FCCM 2022 – May 18, 2022

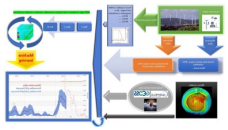
Designing memory architectures with high-level synthesis: What could possibly go wrong?

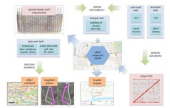
Christian Pilato

Assistant Professor

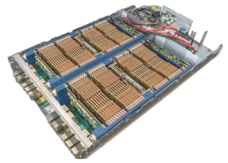
christian.pilato@polimi.it

The EVEREST Project





Big data applicati
heterogeneous d




- App designers are not FPGA experts
- Hardware accelerators require many
- an have different

H2020 Project – 10 partners, 6 countries

Project Coordinator: Christoph Hagleitner, IBM Research Europe

Scientific Coordinator:

Budget: ~5M€

Start date: October 1, 2020

EVEREST

Compilation

Unified hardware gene
(high-level synth

MLIR

Generation of **variants**

Increase quality of accelerators

Improve applications' results

Runtime

daptation to variants

Virtualization of resources

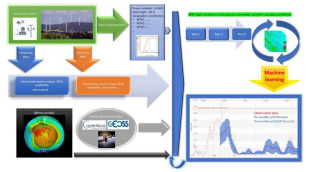
Multi-node support

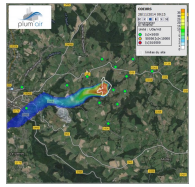
POLITECNICO MILANO 1863

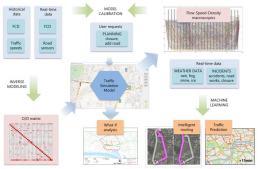
EVEREST Approach

Big data applications with heterogeneous data sources

Three use cases







What are the relevant requirements for data, languages and applications?

How to design data-driven policies for computation, communication, and storage?

How to create FPGA accelerators and associated binaries?

How to manage the system at runtime?

How to evaluate the results?

How to disseminate and exploit the results?

EVEREST SDK

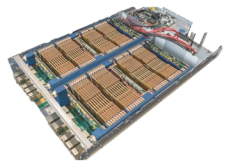
Open-source framework to support the optimization of selected workflow tasks

FPGA-based architectures to accelerate selected kernels

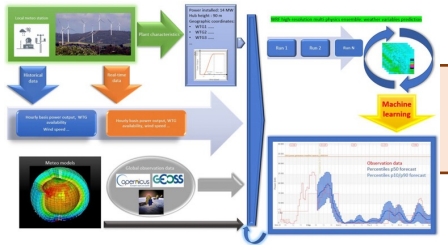
CPU-based infrastructure

+

Two FPGA-based clusters

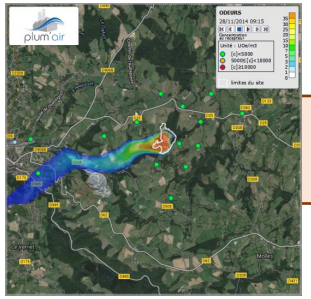


EVEREST Use Cases



Renewable energy production prediction

★ Improve **quality of the predictions**

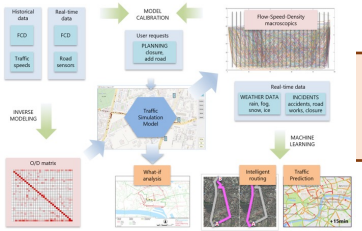


Air-quality monitoring of industrial sites

★ Improve the **response time of predictions**

Weather prediction modelling (WRF)

★ Accelerate kernels to execute more tests



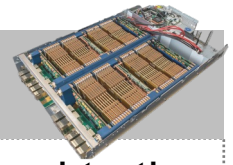
Traffic modeling for intelligent transportation

★ Improve the **overall performance of traffic simulation**

Accelerated computationally-intensive kernels + **Machine-learning** kernels

EVEREST Target System

cloudFPGA



- **Disaggregated FPGAs** directly attached to the network (64 FPGA instances)
- **Low latency** and **high bandwidth** system
- Separation between **Shell** and **Role** modules
- **cFDK framework** for system generation

FPGA-Accelerated HPC Cluster

- Cluster of **PCIe-attached FPGAs** (Alveo) with HBM architecture (up to 460 GB/s per board)
- **Xilinx Vitis framework** for HLS and system integration
- Support for the integration of **custom HDL**

CPU Reference System

- CPU-based infrastructure to **execute end-to-end workflows**, **manage storage**, and **data transfers**
- Extended to support the **offloading of tasks** to **FPGA servers**

LEXIS

Exploit **spatial parallelism**

High **memory bandwidth**

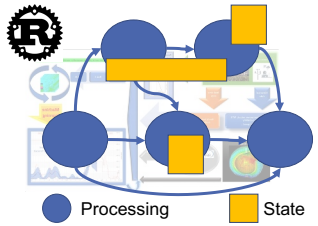
Different nodes to better **match applications**

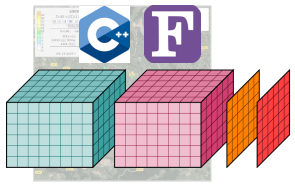
Data-intensive (memory-bound) applications

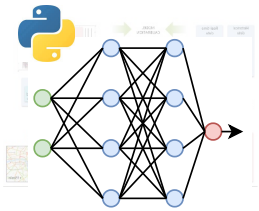
Seamless support for multiple nodes

Limited **FPGA resources** (esp. memories)

EVEREST System Development Kit (SDK)

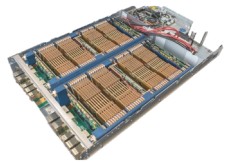






Different **input flows**
starting from different **input languages**

Support for **multiple target boards**



Collection of interoperable and open-source tools to create hardware/software systems that can adapt to the target system, the application workflow, and the data characteristics

- Compilation framework based on **MLIR** to unify the input languages
- **High-level synthesis** and **hardware generation flow** to automatically create optimized architectures
- Creation of hardware and software variants to match architecture features
- Use of state-of-the-art frameworks and commercial toolchains for FPGA synthesis

tvm
MLIR
DandA
XILINX VITIS
cFDK
HEAppE
HyperLoom
(and more...)

The Case of Computational Fluid Dynamics

Numerical simulations are becoming more and more popular for many applications

- **Computational Fluid Dynamics** (CFD) is a representative application that requires to solve partial differential equations
- Kernel is the **Inverse Helmholtz operator** (parametric with respect to polynomial degree *p*) – "Helmholtz" for the friends

```
1 var input S   : [11 11]
2 var input D   : [11 11 11]
3 var input u   : [11 11 11]
4 var output v  : [11 11 11]
5 var t         : [11 11 11]
6 var r         : [11 11 11]
7 t = S # S # S # u . [[1 6] [3 7] [5 8]]
8 r = D * t
9 v = S # S # S # t . [[0 6] [2 7] [4 8]]
```

Final result is obtained by **"small" contributions** on **independent data**

- CFD kernel is composed of **three high-level tensor operators** (two contractions and one Hadamard product) repeated millions of times – good for spatial parallelism
- Each operator requires $p^2 + 2 \cdot p^3$ *(double) elements* as input and produces p^3 *(double) elements* – 21.74 KB + 10.40 KB per element when p = 11
- It requires additional *six tensors* (p^3 *elements)* to store intermediate results – additional 62.39 KB

Hardware HPC (Memory) Architectures

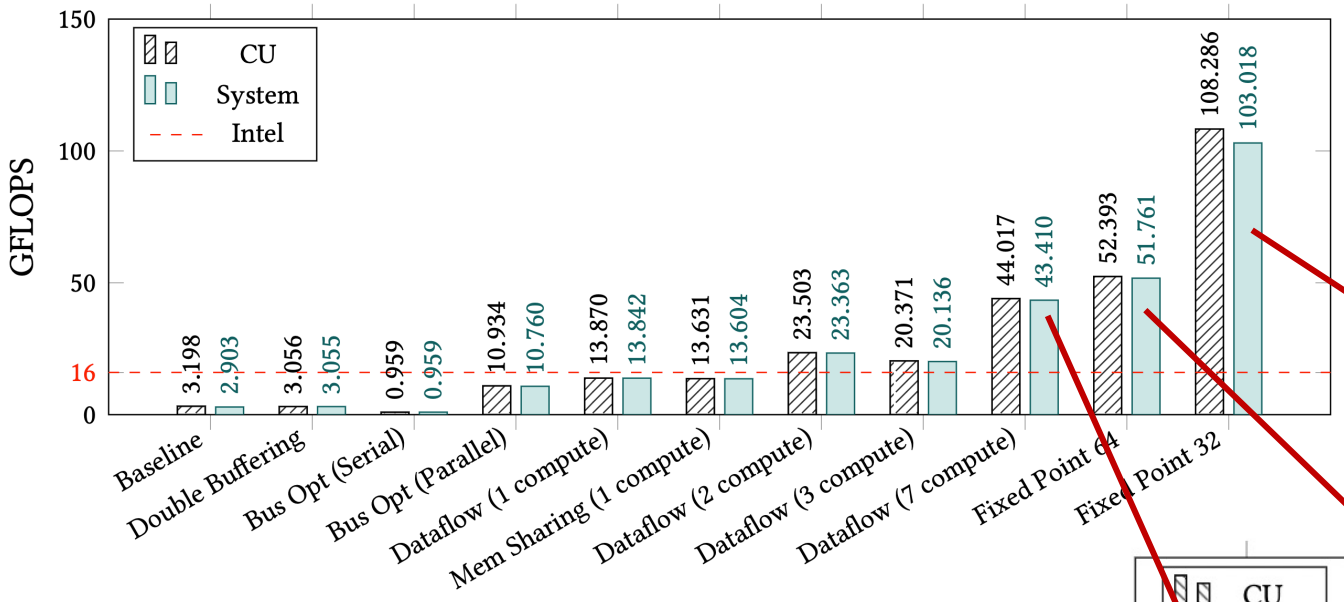
What do we mean with **memory architecture**?

Every hardware module that is responsible to provide data to the accelerator kernels

Additional issues:

- BRAM resources are limited
  - Helmholtz operator requires >94 KB of local data
  - If local storage is not optimized, the number of parallel kernels can be limited
- Application-specific details can be used to optimize the data transfers
  - In Helmholtz, one of the tensors is constant over all elements – how to match these details with platform characteristics?
  - Better to transfer data for a “batch” of elements and then execute them in series – how many? again, limited storage

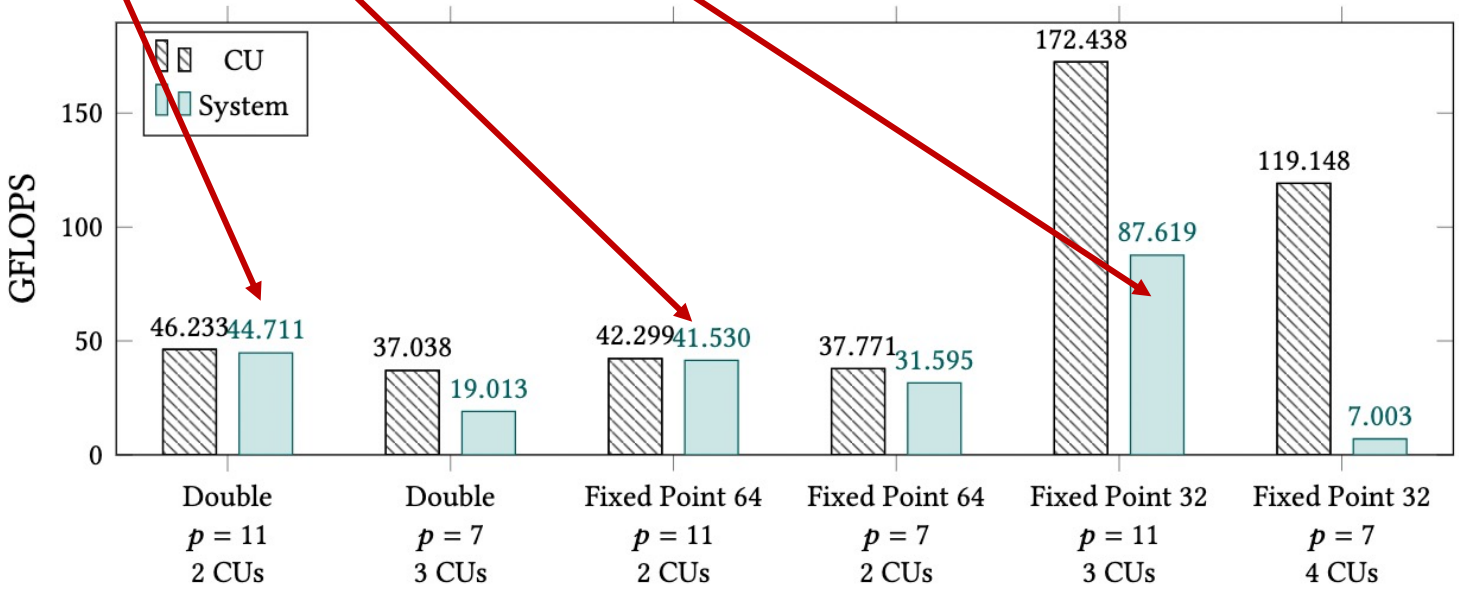
Results on HBM FPGA – Spoiler Alert!



S. Soldavini, K. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillón, C. Pilato:. "Automatic Creation of High-Bandwidth Memory Architectures from Domain-Specific Languages: The Case of Computational Fluid Dynamics" arXiv'22

Best performance: 103 GOPS
(118x faster than our "starting point")

????



Results are 6x better than Intel ones [optimized, vectorized implementation] (~25x more energy efficient)

Possibility of integrating **custom data formats** and configure memories and data transfers accordingly

Challenges for HPC Architectures (i)

We can identify common challenges to most of the FPGA-based HPC architectures (e.g., network-attached cloudFPGA or bus-attached Alveo)

Challenge 1: Input languages and frameworks

- Application designers are usually not FPGA experts and may use high-level framework that are not supported by current HLS tools – how to talk with them?

Challenge 2: CPU-Host Communication Cost

- FPGA logic requires the data on the board, but data transfers can be much more expensive than kernel computation – how to execute more than one point?

Challenge 3: Read/Write Burst Transactions

- We need to determine the proper size of the transactions to get the maximum performance – how to reorganize the data transfers and get the parameters?

Challenges for HPC Architectures (ii)

Challenge 4: Full Bandwidth Utilization

- AXI interfaces may be large (e.g., 256 bits on the Alveo) – how to leverage them?
- HBM architectures have many channels - how to parallelize data transfers?

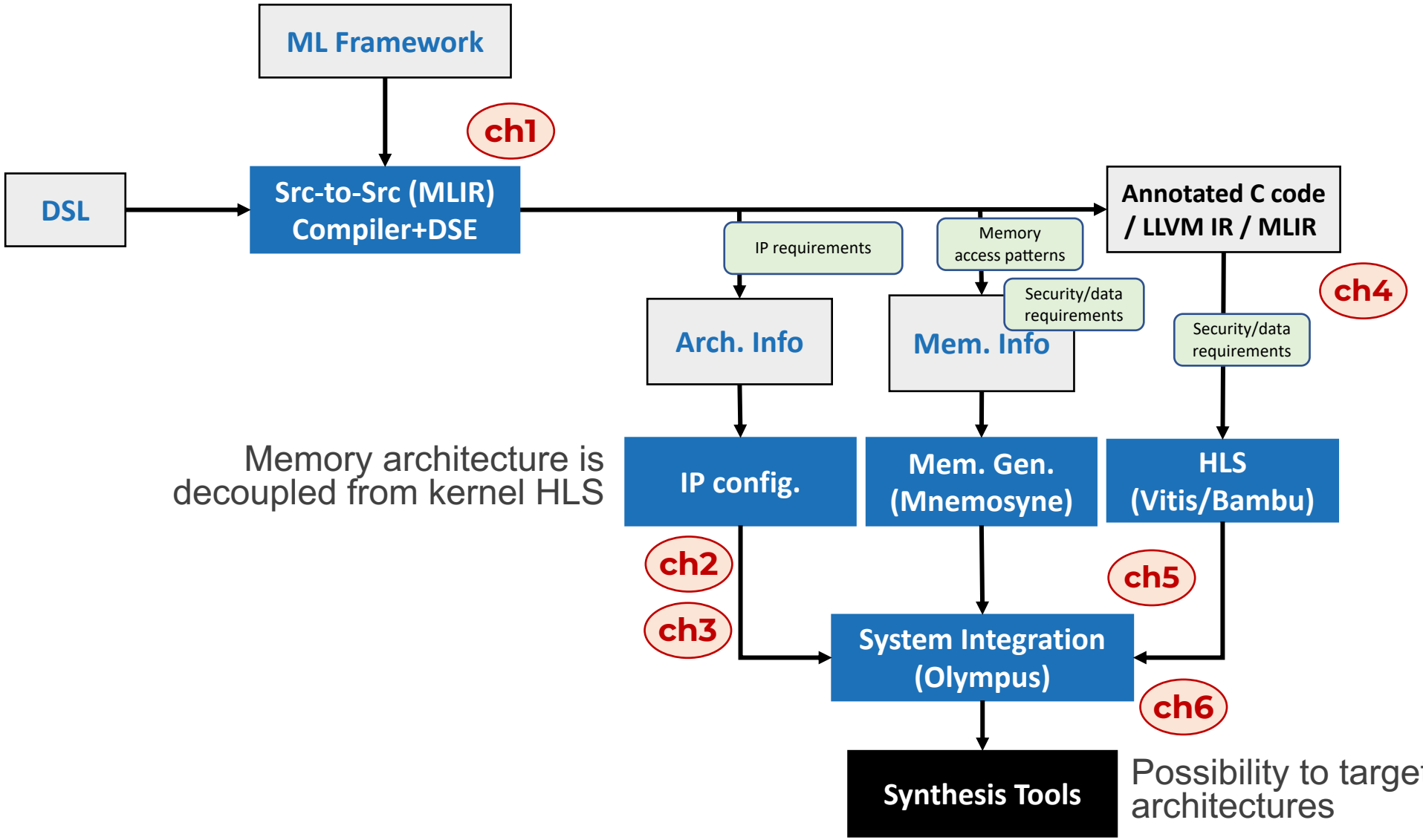
Challenge 5: Data Allocation

- Data must be placed in memory to maximize its utilization but also to enable efficient data transfers/computation –how to use custom data layouts?

Challenge 6: Synthesis-Related Issues

- FPGA devices are large but still not sufficient for hosting many kernels – how to trade-off optimizations and parallel instances?
- FPGA (or architectures) may be different – how to separate platform-agnostic and platform-dependent parts?
- FPGA logic architectures are complex and may introduce performance degradation – how to "guide" the synthesis process?

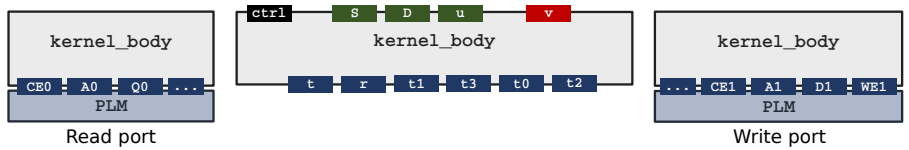
Hardware Compilation Flow

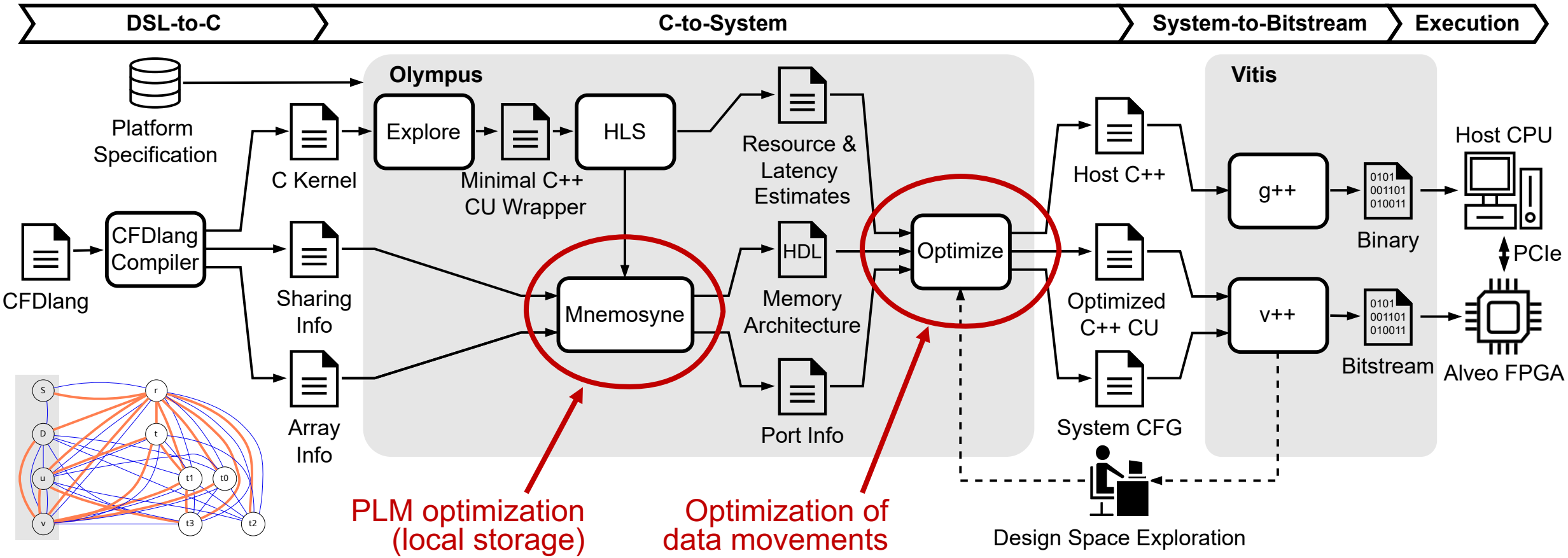


From DSL to Bitstream – Focus on Memory

```
var input S    : [11 11]
var input D    : [11 11 11]
var input u    : [11 11 11]
var output v   : [11 11 11]
var t          : [11 11 11]
var r          : [11 11 11]
t = S # S # S # u . [[1 6] [3 7] [5 8]]
r = D * t
v = S # S # S # t . [[0 6] [2 7] [4 8]]
```

```
void kernel_body(double S[11][11], double D[11][11][11], double u[11][11][11],
                 double v[11][11][11],
                 double t[11][11][11], double r[11][11][11], double t1[11][11][11],
                 double t3[11][11][11], double t0[11][11][11], double t2[11][11][11])
```





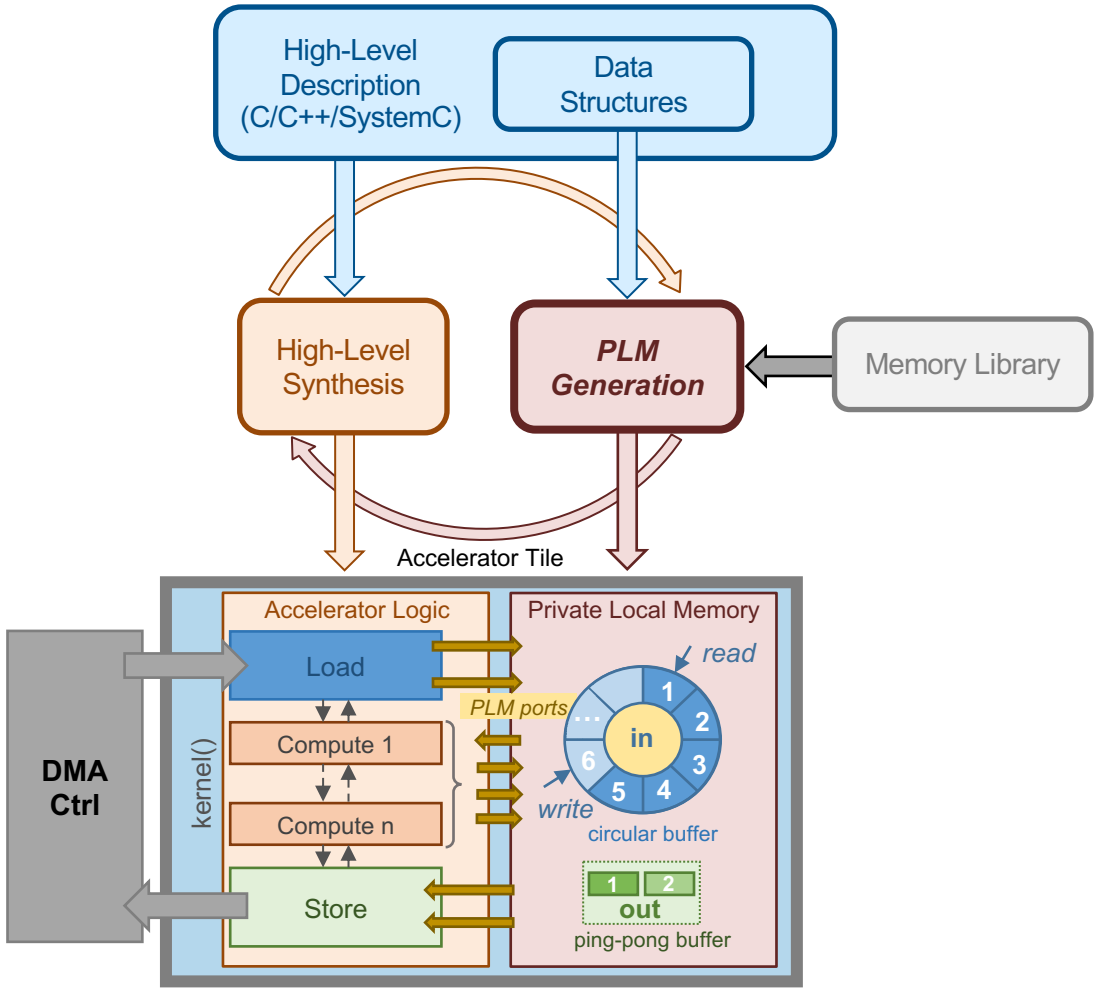
PLM Customization for Heterogeneous SoCs

High-Level Synthesis (HLS) to create the accelerator logic

- Definition of memory-related parameters (e.g. number of process interfaces)

Generation of **specialized PLMs**

- Technology-related optimizations
- Possibility of system-level optimizations across accelerators



Reuse What is not Used

Generally, we use one **PLM unit** (possibly composed of many banks) for each **data structure** (array)

Reuse the same memory IPs
for several data structures

“Two data structures are compatible if they can be allocated to the same PLM unit (memory IPs)”

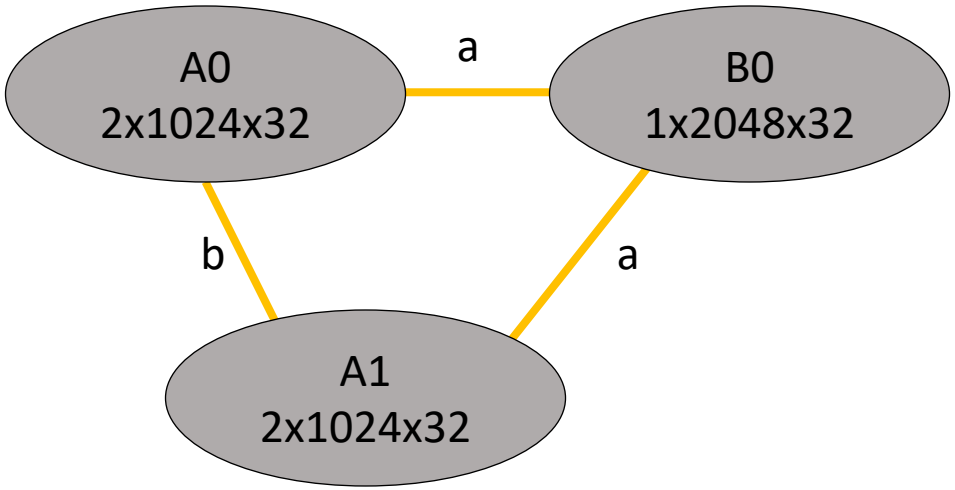
A common case: accelerator parts never executed at the same time

- Possible only at system-level, when integrating the components
- Optimizations of accelerator logic and memory subsystem are independent

Memory Compatibility Graph (MCG)

Graph to represent the possibilities for optimizing the data structures

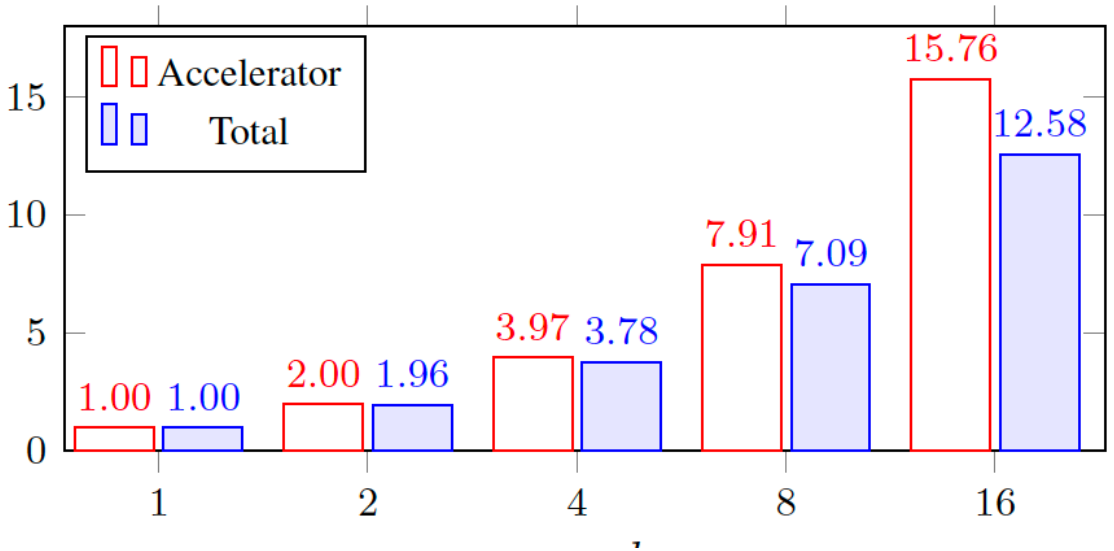
- Each node represents a data structure to be allocated, annotated with its data footprint (after data allocation)
- Each edge represents compatibility between the two data structures
- Can be automatically extracted from the MLIR-based compiler flow
  - Variant exploration to achieve the "best solutions"

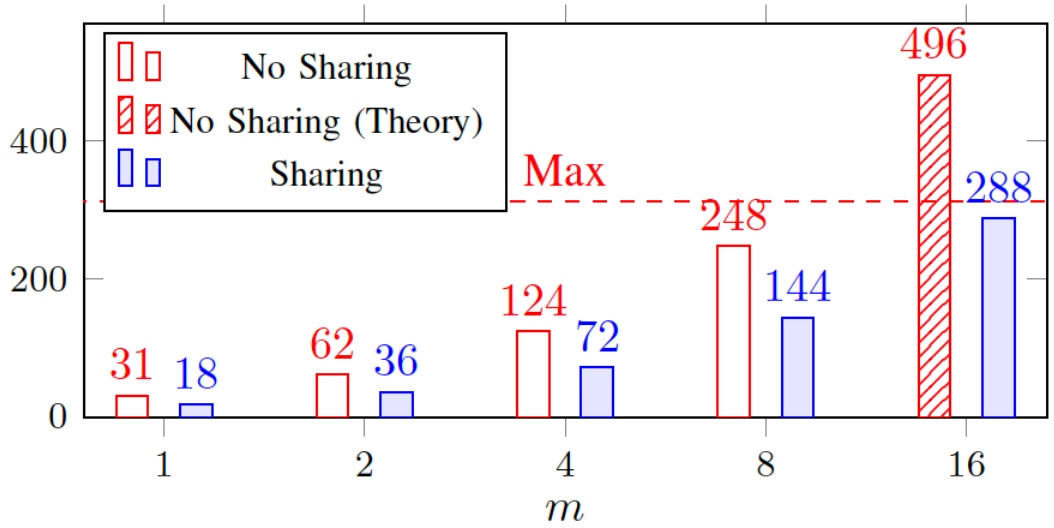


a) **Address-space compatibility**: the data structures are compatible and can use the same memory IPs
b) **Memory-interface compatibility**: the ports are never accessed at the same time and the data structures can stay in the same memory IP

Preliminary Evaluation

- Xilinx Zynq UltraScale+ MPSoC ZCU106 board
  - CFD simulation of 50,000 elements
- Memory sharing allows us to fit more kernels



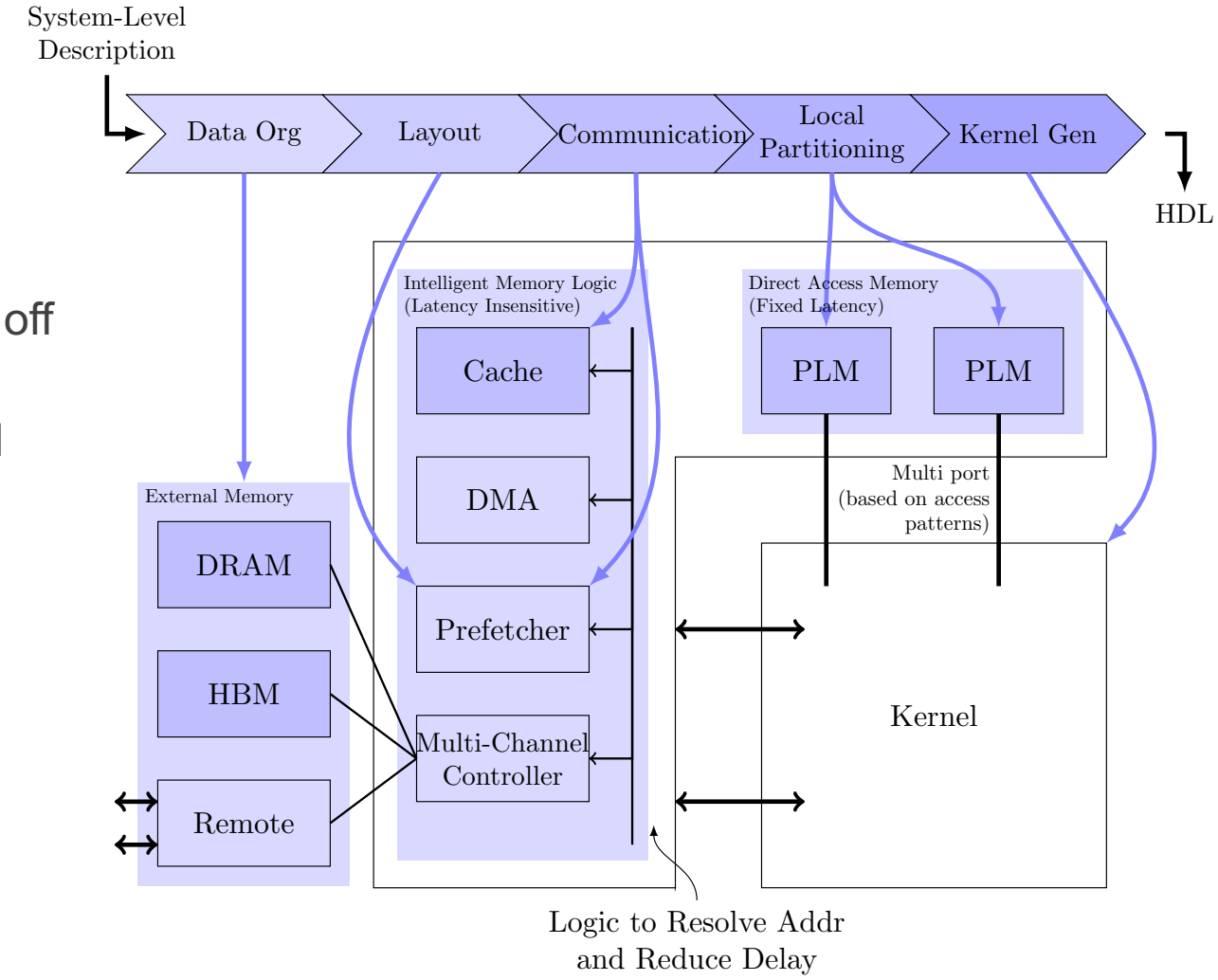


	m, k	LUT		FF		DSP	
No Sharing	1	11,318	(4.9%)	9,523	(2.1%)	15	(0.9%)
	2	15,929	(6.9%)	12,583	(2.7%)	30	(1.7%)
	4	25,728	(11.2%)	18,663	(4.1%)	60	(3.5%)
	8	42,679	(18.5%)	30,795	(6.7%)	120	(6.9%)
Sharing	1	11,292	(4.9%)	9,533	(2.1%)	15	(0.9%)
	2	15,572	(6.8%)	12,596	(2.7%)	30	(1.7%)
	4	24,480	(10.6%)	18,663	(4.1%)	60	(3.5%)
	8	42,141	(18.3%)	30,782	(6.7%)	120	(6.9%)
	16	77,235	(33.5%)	55,053	(12.0%)	240	(13.9%)

Next Step: Let's Put Memory First

We are building an MLIR **compilation flow** for **automatic memory specialization**:

- **MLIR Input** – DSL description of the system functionality
- **Data Organization** – Determine which data resides off chip (also based on user/compiler annotations)
- **Layout** – Reorganize communication to exploit local memories and perform efficient parallel computation
- **Communication** – Configure prefetcher to hide transfer latency
- **Local Partitioning** – Determine multi-bank PLM architecture (Mnemosyne)
- **HLS** – Generate computation part (interfacing with existing HLS tools, e.g., open-source Vitis HLS frontend)
- **HDL Output** – Automated code generation and system-level integration based on the target platform



S. Soldavini and C. Pilato. "Compiler Infrastructure for Specializing Domain-Specific Memory Templates" LATTE'21

Olympus – Automated System-Level Integration

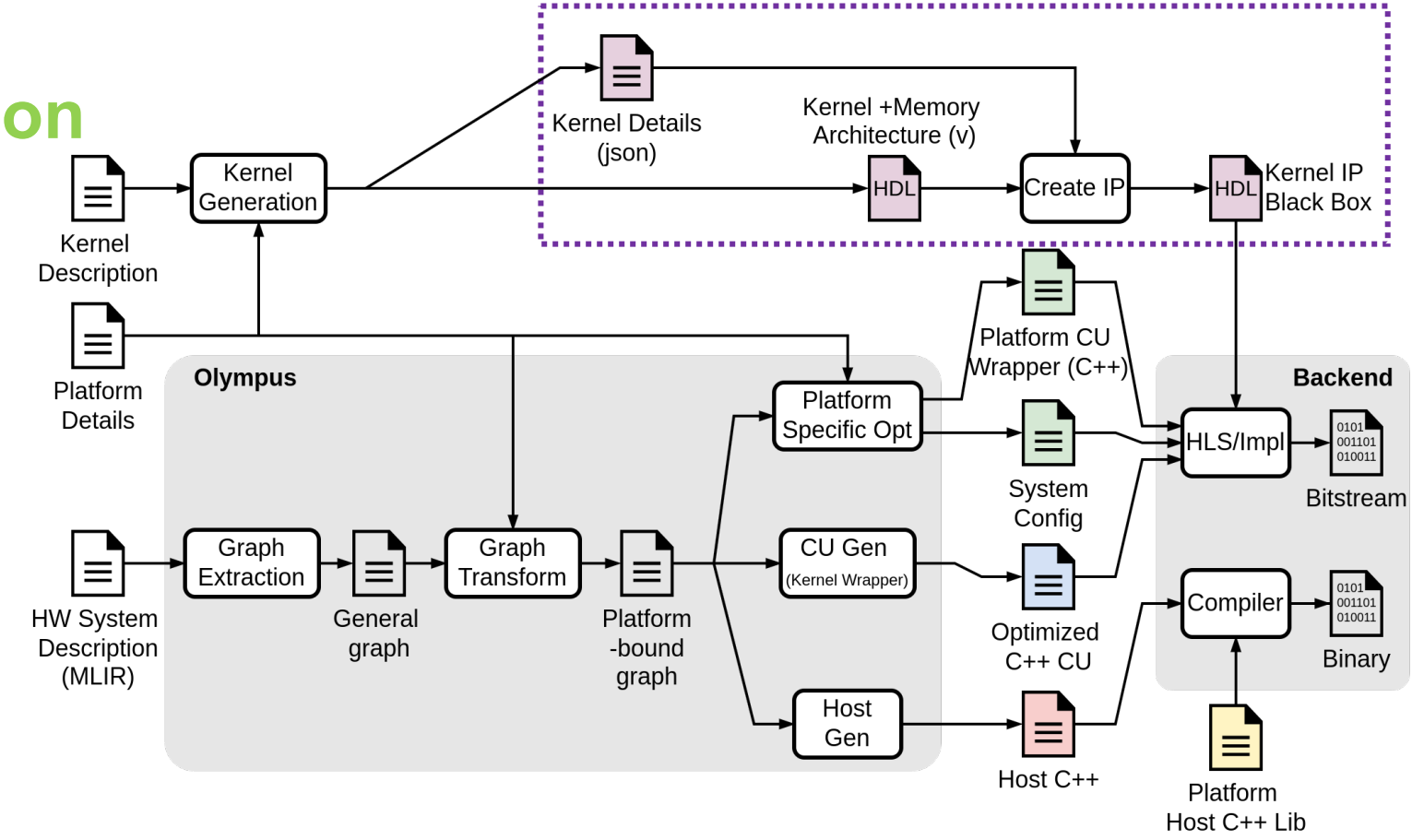
We are developing a complete hardware architecture generation flow based on **MLIR description** of the **system functionality**

Platform-specific description

- HBM-based Xilinx Alveo
- IBM CloudFPGA
- …

Host code generation

- Based on **platform libraries** for the specific target



Conclusions

Data management optimizations are becoming the key for the creation of **efficient FPGA architectures** (… more than pure kernel optimizations)

HLS is now used not only to create accelerator kernels but also to generate the **system-level architecture**

- **Portable solutions** across multiple target platforms

Novel **HBM architectures** offer high bandwidth (that's why they are called *high-bandwidth memory* architectures… ☺) but their design is complex:

- Necessary to match **application requirements** and **technology characteristics**
- We propose an **MLIR-based compilation flow** that directly interfaces with **commercial HLS tools**

Work done in collaboration with Stephanie Soldavini (Politecnico di Milano), Mattia Tibaldi (Politecnico di Milano), Jeronimo Castrillon (TU Dresden), Karl F. A. Friebel (TU Dresden), and Gerald Hempel (TU Dresden)

Thank you!

Christian Pilato, christian.pilato@polimi.it

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957269